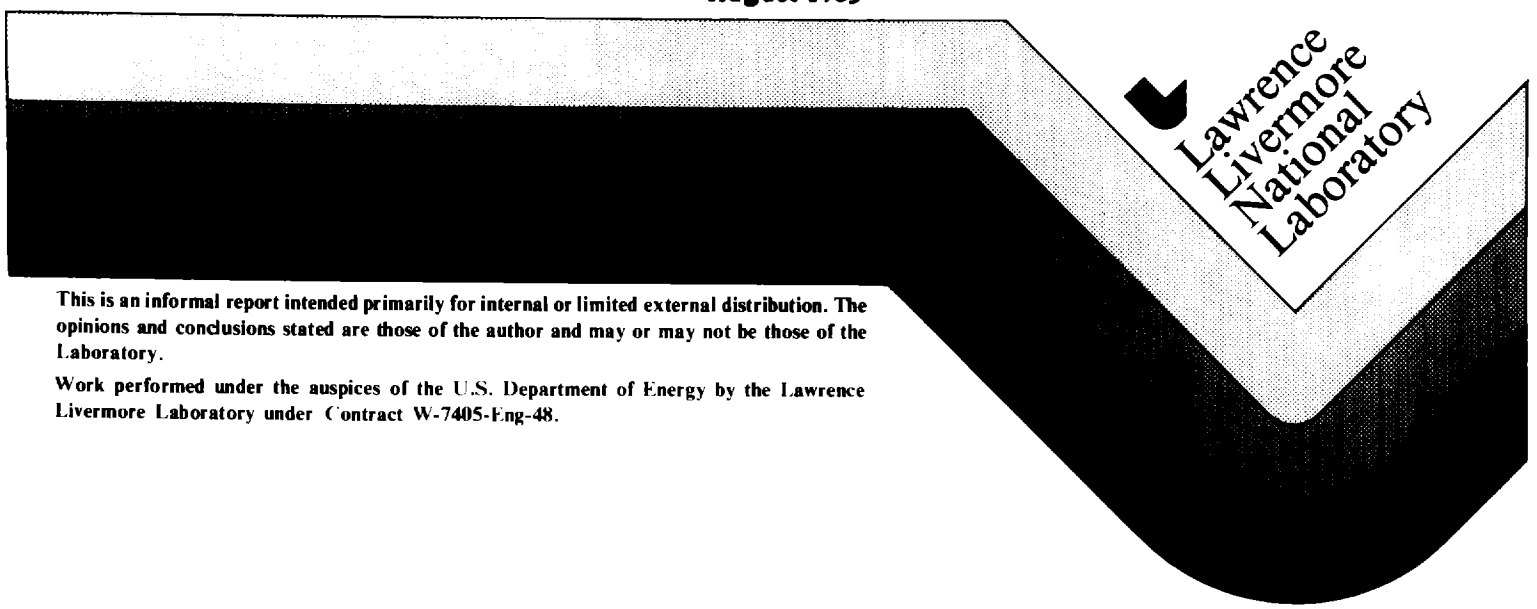
UCID- 20539

POSSOL Poisson Equation Solver

William J. Orvis

August 1985



Work performed under the auspices of the U.S. Department of Energy by the Lawrence Livermore Laboratory under Contract W-7405-Eng-48.

DISCLAIMER

Printed in the United States of America
Available from
National Technical Information Service
U.S. Department of Commerce
5285 Port Royal Road
Springfield, VA 22161
Price: Printed Copy $; Microfiche $4.50

Page Range	Domestic Price	Page Range	Domestic Price
001-025	$ 7.00	326-350	$ 26.50
026-050	8.50	351-375	28.00
051-075	10.00	376-400	29.50
076-100	11.50	401-426	31.00
101-125	13.00	427-450	32.50
126-150	14.50	451-475	34.00
151-175	16.00	476-500	35.50
176-200	17.50	501-525	37.00
201-225	19.00	526-550	38.50
226-250	20.50	551-575	40.00
251-275	22.00	576-600	41.50
276-300	23.50	601-up[1]	
301-325	25.00		

[1]Add 1.50 for each additional 25 page increment, or portion thereof from 601 pages up.

POSSOL Poisson Equation Solver

by

William J. Orvis

ABSTRACT

POSSOL is a two-dimensional Poisson solver for problems with arbitrary non-uniform gridding in cartesian coordinates. Actually, it will solve the Helmholtz equation on an arbitrary, non-uniform grid on a rectangular domain with unmixed boundary conditions. The routine is similar to PWSCRT developed by Schwarztrauber and Sweet at the National Center for Atmospheric Research. The routine is also amenable to the capacitance matrix technique which can be used to solve problems with mixed boundary conditions.

INTRODUCTION

In 1975, Schwarztrauber and Sweet at the National Center for Atmospheric Research (NCAR) wrote a set of FORTRAN subroutines for solving the Helmholtz equation in two dimensions[1]. These routines all use the Buneman variant of cyclic reduction to solve the standard, five point difference approximation in several different coordinate systems:

Routine	Coordinate system
PWSCRT	Cartesian
PWSPLR	Polar
PWSCYL	Cylindrical
PWSCSP	Spherical with axisymmetry
PWSSSP	Spherical on the unit sphere

All of these routines require a uniform mesh and unmixed boundary conditions. For my semiconductor device modeling work, I needed to solve the Poisson equation on a non-uniform mesh in cartesian coordinates. As a result, I have created the subroutine

POSSOL which is similar to PWSCRT, but allows a non-uniform mesh. I also needed to be able to have mixed boundary conditions on any side of the problem, but the routines listed above (including POSSOL) only allow one type of boundary condition on any one side. In order to handle more than one type of boundary condition on a side, I developed a form of the capacitance matrix technique around POSSOL.

METHOD

POSSOL

The five routines mentioned above solve the Helmholtz equation in two dimensions.

$$\nabla^2 U + \lambda U = f \qquad (1)$$

using a five point difference approximation. Four of the five subroutines (PWSCRT, PWSPLR, PWSCYL and PWSSSP) solve this approximation with the routine POIS, which solves the following linear system of equations,

$$A(i)*U(i-1,j) + B(i)*U(i,j) + C(i)*U(i+1,j) +$$
$$U(i,j-1) - 2.*U(i,j) + U(i,j+1) = f(i,j). \qquad (2)$$

where A, B and C are constant coefficient arrays.

The other routine (PWSCSP) uses the routine BLKTRI which solves the following slightly different set of linear equations,

$$AN(j)*U(i,j-1) + AM(i)*U(i-1,j) + (BN(j) + BM(i)) * U(i,j) +$$
$$CN(j)*U(i,j+1) + CM(i)*U(i+1,j) = f(i,j). \qquad (3)$$

where AN, AM, BN, BM, CN and CM are constant, coefficient arrays. In both cases, i ranges from 1 to m+1 and j ranges from 1 to n+1, where m and n are the number of panels in the x and y directions (i.e. there are m+1 and n+1 grid points in the x and y directions respectively).

The routine PWSCRT uses the routine POIS to solve the Helmholtz equation in cartesian coordinates on a uniform mesh. The x coordinate ranges from A to B and the y coordinate ranges from C to D. The Helmholtz equation in two-dimensional cartesian coordinates is,

$$\frac{\partial^2 U}{\partial x^2} + \frac{\partial^2 U}{\partial y^2} + \lambda U = f \qquad (4)$$

This equation is discretized on the uniform mesh:

$$\Delta x = (B - A)/m$$

$$\Delta y = (D - C)/n$$

as,

$$(1/\Delta x) * (U(i+1,j) - 2*U(i,j) + U(i-1,j)) +$$

$$(1/\Delta y) * (U(i,j+1) - 2*U(i,j) + U(i,j-1)) + \lambda * U(i,j) = F(i,j) \qquad (5)$$

These equations can easily be put into the form required by POIS (equation 2 above) by multiplying through with Δy.

However, for my problem, Δx and Δy are functions of x and y respectively, which results in a much more complicated discrete equation. The non-uniform mesh is devfined with:

$$\Delta x^+ = x(i+1) - x(i)$$
$$\Delta x^- = x(i) - x(i-1)$$
$$\Delta x = (\Delta x^+ + \Delta x^-)/2$$

$$\Delta y^+ = y(j+1) - y(j)$$
$$\Delta y^- = y(j) - y(j-1)$$
$$\Delta y = (\Delta y^+ + \Delta y^-)/2$$

and the discrete equation is:

$$\frac{1}{\Delta x}\left[\frac{U(i+1,j)-U(i,j)}{\Delta x^+} - \frac{U(i,j)-U(i-1,j)}{\Delta x^-}\right] +$$

$$\frac{1}{\Delta y}\left[\frac{U(i,j+1)-U(i,j)}{\Delta y^+} - \frac{U(i,j)-U(i,j-1)}{\Delta y^-}\right] + \lambda * U(i,j) = f(i,j) \qquad (6)$$

Now, this equation can not be put into a form that POIS can use, but is already in a form that can be solved by BLKTRI. The constants in equation 3 are defined with,

$$AM(i) = 1/(\Delta x^- * \Delta x)$$
$$BM(i) = -2/(\Delta x^+ * \Delta x^-) + \lambda$$
$$CM(i) = 1/(\Delta x^+ * \Delta x)$$

$$AN(j) = 1/(\Delta y^- * \Delta y)$$
$$BN(j) = -2/(\Delta y^+ * \Delta y^-)$$
$$CN(j) = 1/(\Delta y^+ * \Delta y) \quad (7)$$

Note that λ could have been added to BM or BN with equal results.

Now, the boundary conditions need to be added by modifying the right or left sides of those equations, defined by equation 3, that involve grid points along the boundaries. This must be done because BLKTRI is only an equation solver, and does not autoimatically insert boundary conditions. There are three possible types of boundary conditions that can be used with this routine: fixed, derivative or periodic. For example, consider the one-dimensional equation with $U(i)=U_0$ for $i = 0$. In this problem, the equation at $i = 1$ is:

$$AN(1)*U_0 + BN(1)*U(1) + CN(1)*U(2) = F(1) \quad (8)$$

which we reorganize to:

$$BN(1)*U(1) + CN(1)*U(2) = F(1) - AN(1)*U_0 \quad (9)$$

We then redefine AN(1) and F(1),

$$F(1) = F(1) - AN(1)*U_0$$
$$AN(1) = 0 \quad (10)$$

before calling BLKTRI. Note that they must be redefined in the order shown above.

For derivative boundary conditions, at $i = 1$, of the form:

$$\frac{\partial U}{\partial x} = BDA \quad (11)$$

where BDA is the value of the derivative. We first finite difference equation (11) at the point $i = 1$,

$$(U(2) - U(0))/(2*\Delta x) = BDA \quad (12)$$

Next, we solve it for U(0) and insert it into the problem equation(a one-dimensional version of equation 3) at $i = 1$,

$$AN(1)*(U(2)-2*\Delta x*BDA) + BN(1)*U(1) + CN(1)*U(2) = F(1) \quad (13)$$

Equation 13 is then reorganized into,

$$BN(1)*U(1) + (AN(1)+CN(1))*U(2) = F(1) +2*\Delta x*BDA*AN(1) \quad (14)$$

and, as before, we then redefine the terms CN(1), F(1) , and AN(1) (in that order):

$$\begin{aligned} CN(1) &= CN(1) + AN(1) \\ F(1) &= F(1) + 2* \Delta x*BDA*AN(1) \\ AN(1) &= 0 \end{aligned} \quad (15)$$

before calling BLKTRI.

Finally, for periodic boundary conditions, we assume that U(m) = U(0), U(m+1) = U(1) etc., in which case, AN, BN and CN take on their normal values (equation 7).

For the two-dimensional problems solved by POSSOL, the methods described above are applied to all of the grid points on the four boundaries. In which case, U_0 and BDA become arrays, with one value for each grid point along the boundary i = 0. Six other arrays are defined to hold the values along the other three boundaries.

There is one more situation that must be considered for a general purpose equation solver, and that is the case where λ is zero and the boundary conditions are all either derivative or periodic. In this case a solution may not exist, and if one does, it is not unique (i.e. if U is a solution, $U + U_0$ is also a solution for any constant U_0). If a solution does exist, then the calling program must determine what constant, if any, must be added to the solution. As to the existance of a solution, the following paragraphs will explain how POSSOL treats that problem.

Consider the matrix equation that represents the discrete form of some equation being solved with POSSOL:

$$AU = f \quad (16)$$

where A is a matrix and U and f are vectors. While the original equation from which this set of discrete equations was formed may have had a solution, the process of finite differencing it, plus round-off error may have resulted in a set of equations with no possible solution. In their report, Swarztrauber and Sweet describe a way to correct this situation. They show that equation (16) has a solution if and only if,

$$\mathbf{h}^t\mathbf{f} = 0 \tag{17}$$

for all $\mathbf{h}$ defined by,

$$\mathbf{A}^t\mathbf{h} = 0 \tag{18}$$

Using (17) and (18), we determine a perturbation to f in equation (16) that will insure that it has a solution,

$$p = (\mathbf{h}^t\mathbf{f}/\mathbf{h}^t\mathbf{e})\mathbf{e} \tag{19}$$

where e is the unit vector (1,1,1...).

We then create a new right hand side for equation 16,

$$g = f - p \tag{20}$$

where g is the new right hand side. Note that as long as the perturbation on f is small compared to f, the resulting solution from POSSOL will be correct. If the perturbation is large, then you will end up solving a completely different problem. This usually indicates that your original equaiton does not have a solution or that you have made an error while inputting it. Therefore, the value of the perturbation (p) should be checked after control is returned from POSSOL and compered with the values in the vector f. If p is large compared to f then the calling program should take appropriate action.

For a problem solved with POSSOL, $\mathbf{h}$ can be broken into an x part and a y part as,

$$\mathbf{h} = \mathbf{h}_x{}^t\,\mathbf{h}_y \tag{21}$$

Then, for either of these two components of $\mathbf{h}$ there are two possible combinations of boundary conditions: derivatives specified at both ends, or periodic. For derivative boundary conditions, the value of a component of $\mathbf{h}_x$ (or $\mathbf{h}_y$) can be determined from equation 18 to be ,

$$h_x(k) = (h_x(1)/\Delta x_1)(\Delta x_{k-1} + \Delta x_k) \tag{22}$$

where,

$$\Delta x_k = x(k+1) - x(k)$$

the arguement of h_x refers to the grid point and $h_x(1)$ is arbitrary. This equation can be simplified by letting $h_x(1)$ equal Δx_1 and then inserting for Δx_{k-1} and Δx_k to give,

$$h_x(1) = x(2) - x(1)$$

$$h_x(kmax) = x(kmax) - x(kmax-1)$$

$$h_x(k) = x(k+1) - x(k-1) \qquad (23)$$

where 1 and kmax are the grid points at the boundaries.

The second case has periodic boundary conditions, which gives a slightly different value for h_x. The general expression is,

$$h_x(k) = (h_x(kmax)/(\Delta x_{kmax-1} + \Delta x_{kmax}))(\Delta x_{k-1} + \Delta x_k) \qquad (24)$$

which can be reduced to,

$$h_x(1) = x(2) - x(1) + x(kmax) - x(kmax-1)$$

$$h_x(kmax) = h(1)$$

$$h_x(k) = x(k+1) - x(k-1) \qquad (25)$$

The equations for y follow in exactly the same manner, with the final equation for h following from equation (21),

$$h_{i,j} = h_x(i)\, h_y(j) \qquad (26)$$

Appendix 1 contains the POSSOL interface and subroutine that prepares the problem for solution with BLKTRI.

TESTING

Appendix 2 contains an example problem used to test POSSOL. First, a non-linear grid is mapped onto a rectangular domain. Then, the equation:

$$U = 0.5 + \sin(2*\pi*x/xlen)*\sin(2*\pi*y/ylen) \qquad (27)$$

where xlen and ylen are the length and width of the problem, is inserted into equation 1 and a right hand side (f) is calculated at every point on the grid. This problem was then solved with POSSOL and the result compared with equation 27. The example shown in appendix 2 has fixed boundary conditions along the x boundaries and

derivative boundary conditions along the y boundaries. Several variations of this program were run to test various combinations of fixed, derivative and periodic boundary conditions, and the results were all nearly identical. The only cases where the results were not identical to equation 27 was where there were derivative and/or periodic boundary conditions on all sides, resulting in a problem without a unique solution. The solution returned by POSSOL was centered on zero rather then raised above zero by 0.5 as specified by equation 27. This is not an unexpected result, since any constant may be added to the equaiton, for this case, and still be a solution.

ACCURACY AND TIMING

The errors encountered in the test problems described above had maximum values of about 0.1 % of the analytic solution. According to Schwarztrauber and Sweet, their equation solvers will all give much better accuracy (10^{-8}% on a CDC 7600) for a problem where the derivatives are calculated with the same finite difference equations as are used by the solver rather than using analytic derivatives to calculate the right hand side of the equation. The difference is probably due to errors induced by discretizing the analytic equations.

Timing for POSSOL was not measured, however, it should be similar to that for the routine PWSCSP. Therefore, I have listed the accuracy and timing data for that routine (Table 1). This data is generated by defining random data on an m+1 by n+1 grid; inserting that into a finite differenced form of equation 1 to calculate the right hand side and then solving it with PWSCSP to get the original data back.[1]

TABLE 1 Accuracy and timing for PWSCSP[1]

m	n	Execution Time (msec)	Max. Absolute Difference
32	32	61	$2.1*10^{-12}$
64	64	284	$1.7*10^{-11}$
128	128	1341	$1.1*10^{-10}$

AVAILIBILITY

POSSOL will be available here at LLNL in MSSL (Math and Statistics Software Library). BLKTRI is available from the National Center For Atmospheric Research, Box 3000, Boulder, Colorado. It is also available here in MSSL.

CAPACITANCE MATRIX

The capacitance matrix method is a technique for separating a matrix problem into two parts, which can then be solved separately and then recombined in a manner that gives the correct solution to the original matrix.[2,3] It can be used to allow problems with a mixture of fixed and derivative boundary conditions on an edge to be solved with a solver that only allows one type of boundary condition along an edge. I used it here to separate the discrete form of the Poisson equation with mixed boundary conditions into a part with unmixed boundary conditions that can be solved with POSSOL, and a small sub-matrix that can be solved with a simple matrix solver. These two matrix solutions are then recombined to give the solution to the original problem.

Consider the following matrix equation,

$$Au = f \tag{28}$$

where A is a square m by m matrix and u and f are vectors of length m. Assume now that A consists of two parts: A_2 that is compatable with our matrix solver; and a small part A_1 that is not. We define A as,

$$A = [\, A_1 | A_2] \tag{29}$$

Next we define a matrix B that is solvable with our matrix solver,

$$B = [B_1 | A_2] \tag{30}$$

Note that B consists of the solveable part of A plus another part that has been adjusted to make the whole matrix equation solvable with the matrix solver. What this amounts to with the Poisson equation is taking a side of the problem that is part fixed and part derivative boundary conditions, and rewriting the boundary condition equations to

make the whole side derivative (or fixed) boundary conditions. For example, A_1 could consist of those boundary condition equations that describe the portion of the boundary with fixed boundary conditions, and A_2 would consist of the rest of the problem. B_1 would then be a new set of boundary condition equations describing the same boundary as A_1 but with derivative boundary conditions instead of fixed ones. Using B, we define a new solution vector 0 ,

$$B0 = f \tag{31}$$

and a new matrix C (the capacitance matrix),

$$C = [A_1|0]\ B^{-1}\ [W_1|0] \tag{32}$$

where W_1 is the identity matrix for A_1. Note that the non-zero part of C (C_1) is generally much smaller than A. We define β with the equation,

$$C_1\beta = f_1 - A_1 0 \tag{33}$$

where f_1 is that part of f that goes with A_1, and then solve

$$Bu = f + [W_1|0]\beta \tag{34}$$

to get u, the solution of the original equation.

ACCURACY AND TESTING

Appendix 3 contains an example of an implementation of this method with POSSOL as the main matrix solver. The problem solved is the same as that solved in appendix 2, but with a mixture of derivative and fixed boundary conditions along one side. This problem also requires the general matrix equation solver routine combination DEC/SOL to solve the small capacitance matrix equation (33). DEC does an L/U decomposition of the matrix and SOL back substitutes to get the final solution. The accuracy in this example problem was similar to that for POSSOL alone.

For a single solution, this method should take a little more than twice the amount of time as POSSOL to solve a problem, since it requires two solutions with POSSOL plus a solution of a small matrix with DEC/SOL. For problems that solve Poissons equation

several times on a fixed grid, the first POSSOL solution and the L/U decomposition of the matrix C can be done once and the results saved and reused for each additional solution of the problem. If a large number of solutions are performed, this should approximately cut the time per solution in half.

REFERENCES

1. P. Schwarztrauber, R. Sweet, *Efficient FORTRAN Subprograms for the Solution of Elliptic Partial Differential Equations*, NCAR/TN-109+IA, National Center For Atmospheric Research, Boulder, CO, July, (1975).

2. Private conversation with Dick Hickman Oct (1982).

3. N. K. Madsen, *An Application of a Capacitance Matrix Technique to Expand the Usefulness of a PDE Software Package*, Technical Memorandum No. 77-3, Numerical Mathematics Group, Lawrence Livermore National Laboratory, Livermore, California, Feb., (1977).

APPENDIX 1

```
      subroutine possol (intl,x,n,nbdcnd,bda,bdb,y,m,nbdcnd,bdc,
     1                   bdd,elmbda,f,idimf,pertrb,ierror,w)
c
c ****************************************************
c *
c *   subroutine possol
c *
c *   william j. Orvis   3/12/85
c *   lawrence livermore national laboratory
c *   livermore, california
c *
c * poisson equation solver with a non-uniform grid.
c * this routine is an adaptation of the routine pwscrt by:
c * p. swarztrauber and r. sweet of ncar. it differs from that routine
c * in that it allows a non-uniform grid to be used rather than a
c * uniform grid. the routine blktri from ncar is required.
c * p. schwarztrauber, r. sweet, "efficient fortran subprograms for the
c * solution of elliptic partial differential equations," ncar/tn-109+1a,
c * national center for atmospheric research, boulder, co, july, (1975).
c * if you have a uniform grid, pwscrt will be faster.
c ****************************************************
c
c
c
c
c
c        subroutine possol solves the standard five-point finite
c     difference approximation to the helmholtz equation in cartesian
c     coordinates:
c
c          (d/dx)(d/dx)u + (d/dy)(d/dy)u + lambda*u = f(x,y).
c
c     the arguments are defined as:
c
c     * * * * * * * * * *     on input     * * * * * * * * * *
c
c     intl
c        =0 initial call, quantities that depend on nbdcnd and y are
c           calculated and stored in w before doing a solution.
c        =1 after the initial call, as long as nbdcnd and y dont change
c           between steps. this is about 50% faster than with intl=0.
c
c     x
c       an m+1 dimensional array containing the x grid
c       x(m+1) must be greater than x(1) the boundaries of the grid are:
c       a=x(1) to b=x(m+1).
c
c     m
c       the number of panels into which the interval (a,b) is
c       subdivided.  hence, there will be m+1 grid points in the
c       x-direction given by x(i) for i = 1,2,...,m+1,
c       m must be greater than 4.
c
c     nbdcnd
c       indicates the type of boundary conditions at x = a and x = b.
```

```
c
c       = 0  if the solution is periodic in x, i.e., u(1,j) = u(m+1,j).
c       = 1  if the solution is specified at x = a and x = b.
c       = 2  if the solution is specified at x = a and the derivative of
c            the solution with respect to x is specified at x = b.
c       = 3  if the derivative of the solution with respect to x is
c            specified at x = a and x = b.
c       = 4  if the derivative of the solution with respect to x is
c            specified at x = a and the solution is specified at x = b.
c
c     bda
c       a one-dimensional array of length m+1 that specifies the values
c       of the derivative of the solution with respect to x at x = a.
c       when mbdcnd = 3 or 4,
c
c            bda(j) = (d/dx)u(a,y(j)), j = 1,2,...,m+1  .
c
c       when mbdcnd has any other value, bda is a dummy variable.
c
c     bdb
c       a one-dimensional array of length m+1 that specifies the values
c       of the derivative of the solution with respect to x at x = b.
c       when mbdcnd = 2 or 3,
c
c            bdb(j) = (d/dx)u(b,y(j)), j = 1,2,...,m+1  .
c
c       when mbdcnd has any other value bdb is a dummy variable.
c
c     y
c       an m+1 dimensional array containing the y grid,
c       y(m+1) must be greater than y(1) the boundaries of the grid are:
c       c=y(1) to d=y(m+1).
c
c     n
c       the number of panels into which the interval (c,d) is
c       subdivided.  hence, there will be m+1 grid points in the
c       y-direction given by y(j) for j = 1,2,...,m+1
c       if nbdcnd = 0 then n must be equal to 2**k.
c       if nbdcnd = 1 then n must be equal to 2**k.
c       if nbdcnd = 2 then n must be equal to 2**k-1.
c       if nbdcnd = 3 then n must be equal to 2**k-2.
c       if nbdcnd = 4 then n must be equal to 2**k-1.
c       the operational count of the solver is mnlog2n so make n<m for the
c       best efficiency.
c
c     nbdcnd
c       indicates the type of boundary conditions at y = c and y = d.
c
c       = 0  if the solution is periodic in y, i.e., u(i,1) = u(i,n+1).
c       = 1  if the solution is specified at y = c and y = d.
c       = 2  if the solution is specified at y = c and the derivative of
c            the solution with respect to y is specified at y = d.
c       = 3  if the derivative of the solution with respect to y is
c            specified at y = c and y = d.
c       = 4  if the derivative of the solution with respect to y is
c            specified at y = c and the solution is specified at y = d.
c
c     bdc
```

```
c       a one-dimensional array of length m+1 that specifies the values
c       of the derivative of the solution with respect to y at y = c.
c       when nbdcnd = 3 or 4,
c
c            bdc(i) = (d/dy)u(x(i),c), i = 1,2,...,m+1  .
c
c       when nbdcnd has any other value, bdc is a dummy variable.
c
c     bdd
c       a one-dimensional array of length m+1 that specifies the values
c       of the derivative of the solution with respect to y at y = d.
c       when nbdcnd = 2 or 3,
c
c            bdd(i) = (d/dy)u(x(i),d), i = 1,2,...,m+1  .
c
c       when nbdcnd has any other value, bdd is a dummy variable.
c
c     elmbda
c       the constant lambda in the helmholtz equation.  if
c       lambda .gt. 0, a solution may not exist.  however, poisol will
c       attempt to find a solution.
c
c     f
c       a two-dimensional array which specifies the values of the right
c       side of the helmholtz equation and boundary values (if any).
c       for i = 2,3,...,m and j = 2,3,...,n
c
c            f(i,j) = f(x(i),y(j)).
c
c       on the boundaries f is defined by
c
c            mbdcnd     f(1,j)        f(m+1,j)
c            ------     ---------     --------
c
c              0        f(a,y(j))     f(a,y(j))
c              1        u(a,y(j))     u(b,y(j))
c              2        u(a,y(j))     f(b,y(j))     j = 1,2,...,n+1
c              3        f(a,y(j))     f(b,y(j))
c              4        f(a,y(j))     u(b,y(j))
c
c
c            nbdcnd     f(i,1)        f(i,n+1)
c            ------     ---------     --------
c
c              0        f(x(i),c)     f(x(i),c)
c              1        u(x(i),c)     u(x(i),d)
c              2        u(x(i),c)     f(x(i),d)     i = 1,2,...,m+1
c              3        f(x(i),c)     f(x(i),d)
c              4        f(x(i),c)     u(x(i),d)
c
c       f must be dimensioned at least (m+1)*(n+1).
c
c       note
c
c       if the table calls for both the solution u and the right side f
c       at  a corner then the solution must be specified.
c
c     idimf
```

```
c       the row (or first) dimension of the array f as it appears in the
c       program calling pesrt.  this parameter is used to specify the
c       variable dimension of f.  idimf must be at least m+1  .
c
c     w
c       a one-dimensional array that must be provided by the user for
c       work space.  the length of w must be at least:
c       if nbdcnd=0 2n+log2(n)+m+2+max(4n,6m)+3(n+1)+3*(m+1)
c       if nbdcnd>0 2(m+1)(log2(m+1)-1)+2+max(2n,6m)+3(n+1)+3*(m+1)
c
c
c     **********     on output     **********
c
c     f
c       contains the solution u(i,j) of the finite difference
c       approximation for the grid point (x(i),y(j)), i = 1,2,...,m+1,
c       j = 1,2,...,n+1  .
c
c     pertrb
c       if a combination of periodic or derivative boundary conditions
c       is specified for a poisson equation (lambda = 0), a solution may
c       not exist.  pertrb is a constant, calculated and subtracted from
c       f, which ensures that a solution exists.  pesrt then computes
c       this solution, which is a least squares solution to the original
c       approximation. this solution is not unique and is unnormalized.
c       the value of pertrb should be small compared to the right side
c       f. otherwise , a solution is obtained to an essentially
c       different problem. this comparison should always be made to
c       insure that a meaningful solution has been obtained.
c
c     ierror
c       an error flag that indicates invalid input parameters.  except
c       for numbers 0 and 6, a solution is not attempted.
c
c       = 0  no error.
c       = 1  a .ge. b.
c       = 2  mbdcnd .lt. 0 or mbdcnd .gt. 4  .
c       = 3  c .ge. d.
c       = 5  nbdcnd .lt. 0 or nbdcnd .gt. 4  .
c       = 6  lambda .gt. 0  .
c       = 7  idimf .lt. m+1  .
c       = 8 or 14  m .lt. 5 (from biktri)
c       = 9 or 15  n not in proper form for nbdcnd = 1,2,3,4 (from biktri)
c       = 10 or 16 n not in proper form for nbdcnd = 0 (from biktri)
c       = 11 or 17 biktri failed while computing results that depend on the
c          coefficient arrays w(an1),w(bn1),w(cn1) check these arrays.
c       = 12 or 18 idimf .lt. munk (from biktri)
c       > 13 error is error 6 plus errors 8 to 12.
c
c       since this is the only means of indicating a possibly incorrect
c     call to possol, the user should test ierror after the call.
c     w
c       contains intermediate values that must not be destroyed if
c       pesrt will be called again with intl = 1  .
c
c
c
      dimension       f(idimf,1)
```

```
      dimension       bda(1)     ,bdb(1)     ,bdc(1)     ,bdd(1)     ,
     1                w(1),  x(1),  y(1)
      integer an1,bn1,cn1,am1,bm1,cm1,w1
c
c     check for invalid parameters.
c
      perturb=0
      ierror = 0
      if (x(1).ge.x(m+1)) ierror = 1
      if (mbdcnd.lt.0 .or. mbdcnd.gt.4) ierror = 2
      if (y(1).ge.y(n+1)) ierror = 3
      if (nbdcnd.lt.0 .or. nbdcnd.gt.4) ierror = 5
      if (idimf .lt. m+1) ierror = 7
      if (ierror .ne. 0) return
c
      np1 = n+1
      mp1 = m+1
c
c  set indicies into the w array for an,bn,cn,am,bm,cm and w for blktri
c  these are indicies to the first element of the arrays.
      an1=1
      bn1=an1+np1
      cn1=bn1+np1
      am1=cn1+np1
      bm1=am1+mp1
      cm1=bm1+mp1
      w1=cm1+mp1
c
      np = nbdcnd+1
      mp = mbdcnd+1
c
c  set start and stop limits for y
      goto (100,101,102,103,104),np
  100 nstart=1
      nstop=n
      npp=0
      goto 105
  101 nstart=2
      nstop=n
      npp=1
      goto 105
  102 nstart=2
      nstop=np1
      npp=1
      goto 105
  103 nstart=1
      nstop=np1
      npp=1
      goto 105
  104 nstart=1
      nstop=n
      npp=1
  105 munk=nstop-nstart+1
c
c  set start and stop limits for x
      goto (110,111,112,113,114),mp
  110 mstart=1
      mstop=m
```

```
      app=0
      goto 115
  111 astart=2
      astop=m
      app=1
      goto 115
  112 astart=2
      astop=mp1
      app=1
      goto 115
  113 astart=1
      astop=mp1
      app=1
      goto 115
  114 astart=1
      astop=m
      app=1
  115 munk=astop-astart+1
c
c  fill am,bm,cm
      do 120 i=2,m
      deltaxp=x(i+1)-x(i)
      deltaxm=x(i)-x(i-1)
      deltax=0.5*(deltaxp+deltaxm)
      a(am1+i-1)=1./(deltax*deltaxm)
      a(bm1+i-1)=-2./(deltaxp*deltaxm)+elambda
      a(cm1+i-1)=1./(deltax*deltaxp)
  120 continue
c
c  do boundaries at a
      goto (130,131,131,133,133),ap
c  periodic
  130 deltaxp=x(2)-x(1)
      deltaxm=x(mp1)-x(m)
      deltax=0.5*(deltaxp+deltaxm)
      a(am1)=1./(deltax*deltaxm)
      a(bm1)=-2./(deltaxp*deltaxm)+elambda
      a(cm1)=1./(deltax*deltaxp)
      goto 135
c  specified
  131 do 132 j=nstart,nstop
  132 f(2,j)=f(2,j)-f(1,j)*a(am1+1)
      a(am1+1)=0
      goto 135
c  derivative
  133 deltaxp=x(2)-x(1)
      deltaxm=deltaxp
      deltax=deltaxp
      a(am1)=0
      a(bm1)=-2./(deltaxp*deltaxm)+elambda
      a(cm1)=2./(deltax*deltaxp)
      do 134 j=nstart,nstop
  134 f(1,j)=f(1,j)+bda(j)*deltaxp*a(cm1)
  135 continue
c
c  do boundaries at b
      goto (145,141,143,143,141) ,ap
c specified
```

```
346  141  do 142 j=nstart,nstop
347  142  f(a,j)=f(a,j)-f(ap1,j)*w(ca1+a-1)
348       w(ca1+a-1)=0
349       goto 145
350 c derivative
351  143  deltaxa=x(ap1)-x(a)
352       deltaxp=deltaxa
353       deltax=deltaxa
354       w(ca1+a)=2./(deltax*deltaxa)
355       w(ba1+a)=-2./(deltaxp*deltaxa)+alambda
356       w(ca1+a)=0
357       do 144 j=nstart,nstop
358  144  f(ap1,j)=f(ap1,j)-bdb(j)*deltaxa*w(ca1+a)
359  145  continue
360 c
361 c  fill an,bn,cn
362       do 150 j=2,n
363       deltayp=y(j+1)-y(j)
364       deltaya=y(j)-y(j-1)
365       deltay=0.5*(deltayp+deltaya)
366       w(an1+j-1)=1./(deltay*deltaya)
367       w(bn1+j-1)=-2./(deltayp*deltaya)
368       w(cn1+j-1)=1./(deltay*deltayp)
369  150  continue
370 c
371 c  do boundaries at c
372       goto (160,161,161,163,163),np
373 c  periodic
374  160  deltayp=y(2)-y(1)
375       deltaya=y(np1)-y(n)
376       deltay=0.5*(deltayp+deltaya)
377       w(an1)=1./(deltay*deltaya)
378       w(bn1)=-2./(deltayp*deltaya)
379       w(cn1)=1./(deltay*deltayp)
380       goto 165
381 c  specified
382  161  do 162 i=mstart,mstop
383  162  f(i,2)=f(i,2)-f(i,1)*w(an1+1)
384       w(an1+1)=0
385       goto 165
386 c  derivative
387  163  deltayp=y(2)-y(1)
388       deltaya=deltayp
389       deltay=deltayp
390       w(an1)=0
391       w(bn1)=-2./(deltayp*deltaya)
392       w(cn1)=2./(deltay*deltayp)
393       do 164 i=mstart,mstop
394  164  f(i,1)=f(i,1)+bdc(i)*deltayp*w(cn1)
395  165  continue
396 c
397 c  do boundaries at d
398       goto (175,171,173,173,171) ,np
399 c specified
400  171  do 172 i=mstart,mstop
401  172  f(i,n)=f(i,n)-f(i,np1)*w(cn1+n-1)
402       w(cn1+n-1)=0
403       goto 175
```

```
c derivative
 173  deltaym=y(np1)-y(n)
      deltayp=deltaym
      deltay=deltaym
      a(an1+n)=2./(deltay*deltaym)
      a(bn1+n)=-2./(deltayp*deltaym)
      a(cn1+n)=0
      do 174 i=mstart,mstop
 174  f(i,np1)=f(i,np1)-bdd(i)*deltayp*a(an1+n)
 175  continue
      if (elmbda) 252,240,239
  239 ierror = 6
      go to 252
c
c     for singular problems must adjust data to insure that a solution
c     will exist.
c
  240 if ((nbdcnd.eq.0 .or. nbdcnd.eq.3) .and.
     1    (mbdcnd.eq.0 .or. mbdcnd.eq.3)) ,252
      msp1 = mstart+1
      mstm1 = mstop-1
c
      s1 = 0.
      ht1=0
      do 247 j=nstart,nstop
c
c  sum up the interior points
      s = 0.
      ht=0
      do 242 i=msp1,mstm1
      h=x(i+1)-x(i-1)
      s = s+f(i,j)*h
      ht=ht+h
  242 continue
c
c  add the points at xmin and xmax
c
c  mbdcnd=0
      h1=x(2)-x(1)+x(m+1)-x(m)
      hm=x(m+1)-x(m-1)
      if (mbdcnd.eq.0) 243,
c
c  mbdcnd=3
      h1=x(2)-x(1)
      hm=x(m+1)-x(m)
 243  s=s+f(mstart,j)*h1+f(mstop,j)*hm
      ht=ht+h1+hm
c
c
c  check for values along the y boundary
      if (j.gt.nstart) 244,
      h=y(2)-y(1)
      if (nbdcnd.eq.0) h=h+y(n+1)-y(n)
      goto 246
c
 244  if (j.lt.nstop) 245,
      h=y(n+1)-y(n)
      if (nbdcnd.eq.0) h=y(n+1)-y(n-1)
```

```
      goto 246
c
 245  h=y(j+1)-y(j-1)
 246  s1=s1+s*h
      ht1=ht1+ht*h
 247  continue
      pertrb=s1/ht1
      do 250 j=nstart,nstop
      do 249 i=mstart,mstop
      f(i,j) = f(i,j)-pertrb
  249 continue
  250 continue
c
c  initialize w if required
 252  if (intl.eq.1) 253,
c
      call bliktri (0,
     x         npp,munk,w(an1+nstart-1),w(bn1+nstart-1),w(cn1+nstart-1),
     x         spp,sunk,w(as1+mstart-1),w(bs1+mstart-1),w(cs1+mstart-1),
     x         idimf,f(mstart,nstart),ierr1,w(w1))
c
c  check for an error during the initialization
      if (ierr1.ne.0) 254,
c
c  solve the problem
 253  call bliktri (1,
     x         npp,munk,w(an1+nstart-1),w(bn1+nstart-1),w(cn1+nstart-1),
     x         spp,sunk,w(as1+mstart-1),w(bs1+mstart-1),w(cs1+mstart-1),
     x         idimf,f(mstart,nstart),ierr1,w(w1))
c
c  check for bliktri errors
 254  if (ierr1.eq.0) 256,
      ierror=ierr1+7+ierror
      return
 256  continue
c
c     fill in identical values when have periodic boundary conditions.
c
      if (nbdcnd .ne. 0) go to 260
      do 259 i=mstart,mstop
         f(i,np1) = f(i,1)
  259 continue
  260 if (sbdcnd .ne. 0) go to 262
      do 261 j=nstart,nstop
         f(mp1,j) = f(1,j)
  261 continue
      if (nbdcnd .eq. 0) f(mp1,np1) = f(1,np1)
  262 continue
      return
c     uses subr bliktri
      end
c
```

APPENDIX 2

This is a test problem for the POSSOL routine. It puts a nonuniform grid on a rectangular region with derivative boundary conditions at y=c and y=d, and fixed boundary conditions at x=a and x=b. The correct solution for the region is:

$$U = 0.5 + \sin(2*\pi*x/(b-a))*\sin(2*\pi*y/(d-c))$$

```
 1 c  ***************************************************
 2 c  *
 3 c  *  program postst
 4 c  *
 5 c  *  w. j. orvis  llnl 4/15/85
 6 c  *
 7 c  *  this is a test problem for the possols routine
 8 c  *
 9 c  ***************************************************
10       dimension x(100),y(100),f(100,100),w(1000)
11       dimension bda(100),bdb(100),bdc(100),bdd(100)
12       call dropfile(0)
13       call create(2,"posout",2,-1)
14       call setclose(2,1,1,0)
15       pi=3.141592654
16 c
17 c  n+1 is the arbitrary number of x grid points
18 c  for the best efficiency, set up the problem with n > n
19       n=99
20 c
21 c  set fixed boundary conditions at x=a and x=b
22       nbdcnd=1
23 c
24 c  put in a non-linear x-grid
25       xlen=20.
26       do 100 i=1,n+1
27       x(i)=xlen*sin(pi*float(i-1)/(2.*float(n)))
28  100  continue
29 c
30 c  m+1 is the number of y grids, its value is restricted by nbdcnd
31       m=62
32 c
33 c  set derivative boundary conditions at y=c and y=d
34       nbdcnd=3
35 c
36 c  put in a non-linear y-grid
37       ylen=15.
38       do 200 j=1,m+1
39       y(j)=ylen*sin(pi*float(j-1)/(2.*float(m)))
40  200  continue
41 c
42 c  some constants
43       pix=2.*pi/x(n+1)
```

```
      piy=2.*pi/y(n+1)
c
c  set lambda=0 for this problem
      elmbda=0
c
c  set idimf equal to the first dimension of f in the dimension
c  statement above.
      idimf=100
c
c  set f
      do 300 i=1,m+1
      do 300 j=1,n+1
      f(i,j)=-(pix**2+piy**2)*sin(piy*y(j))*sin(pix*x(i))
  300 continue
c
c  put in the values of the derivatives at y=c and y=d
      do 320 i=1,m+1
      bdc(i)=piy*sin(pix*x(i))
      bdd(i)=bdc(i)
  320 continue
c
c  put the value of the fixed boundaries at x=a and x=b
      do 340 j=1,n+1
      f(1,j)=5.
      f(m+1,j)=5.
  340 continue
c
c  solve the problem
      call possol (0,x,m,mbdcnd,bda,bdb,y,n,nbdcnd,bdc,
     1                  bdd,elmbda,f,idimf,pertrb,ierror,w)
c
c  ierror should be tested for problems with the solution
      write (2,500) ierror
  500 format("ierror after solution = ",i5)
c
c  for singular problems, test pertrb to be sure that
c  it is small compared to f .
      write (2,510) pertrb
  510 format("pertrb = ",e12.5)
c
c  print out the results
      do 600 i=1,m+1
      write (2,550) (f(i,j),j=1,n+1)
  550 format (4e12.5)
  600 continue
c
c  print out the difference between the solution and the correct value
      write (2,650)
  650 format("error values")
      do 800 i=1,m+1
      write (2,550) ((f(i,j)-5.-sin(pix*x(i))*sin(piy*y(j))),j=1,n+1)
  800 continue
c
      call exit(0)
      end
```

APPENDIX 3

This is the same problem as that solved in Appendix 2, except that the boundary conditions are now mixed along the side x=a. The boundary conditions are fixed from grid points 1 through 10 and are derivative from grid points 11 through 63.

```
c  ************************************************
c  *
c  *  program capstx
c  *
c  *  w. j. orvis  llnl 4/15/85
c  *
c  *  this is a test problem for the possols routine
c  *  and the capacitance matrix method. the troublesome boundary is
c  *  at x=a.
c  *
c  ************************************************
      dimension x(100),y(100),f(100,100),w(10000),c(20,20),ip(20)
      dimension bda(100),bdb(100),bdc(100),bdd(100),beta(20)
      call dropfile(0)
      call create(2,"capout",2,-1)
      call setclose(2,1,1,0)
      pi=3.141592654
c
c  m+1 is the arbitrary number of x grid points
c  for the best efficiency, set up the problem with m > n
      m=90
c
c  n+1 is the number of y grids, its value is restricted by nbdcnd
      n=62
c
c  put in a non-linear x-grid
      xlen=20.
      do 100 i=1,m+1
      x(i)=xlen*sin(pi*float(i-1)/(2.*float(m)))
  100 continue
c
c  put in a non-linear y-grid
      ylen=15.
      do 200 j=1,n+1
      y(j)=ylen*sin(pi*float(j-1)/(2.*float(n)))
  200 continue
c
c  some constants
      pix=2.*pi/x(m+1)
      piy=2.*pi/y(n+1)
c
c  set lambda=0 for this problem
      elmbda=0
c
c  set idimf equal to the first dimension of f in the dimension
c  statement above.
      idimf=100
c
```

```
c  set fixed bc at x=b and derivative at x=a
      nbdcnd=4
c
c  set derivative boundary conditions at y=c and y=d
      nbdcnd=3
c
c  set all of the derivatives to zero for the calculation of c
      do 240 j=1,n+1
      bda(j)=0
 240  continue
c
      do 220 i=1,m+1
      bdc(i)=0
      bdd(i)=0
 220  continue
c
c  loop over the p points with fixed bc at x=1
      jpi=1
      jpf=10
      do 510 k=jpi,jpf
c
c  zero all but the k th element of f along i=1
      do 300 i=1,m+1
      do 300 j=1,n+1
      f(i,j)=0
 300  continue
      f(1,k)=1.
c
c  solve b*g(k) = a(k)
      call possol (0,x,m,nbdcnd,bda,bdb,y,n,nbdcnd,bdc,
     1                  bdd,elmbda,f,idimf,pertrb,ierror,w)
c
c  ierror should be tested for problems with the solution
      write (2,500) ierror
 500  format("ierror after solution = ",i5)
c
c  store a column of c
      do 505 j=jpi,jpf
      c(j-jpi+1,k)=f(1,j)
 505  continue
 510  continue
c
c  lu decompose c
      call dec(jpf-jpi+1,20,c,ip,ier)
      write (2,515) ier
 515  format("ier from dec = ",i5)
c
c  begin the main solution sequence
c
c  create v in f
      do 520 i=1,m+1
      do 520 j=1,n+1
      f(i,j)=-(pix**2+piy**2)*sin(pix*x(i))*sin(piy*y(j))
 520  continue
c
c  put in the boundary conditions
c  dont forget the fixed values at x=a
c  set the derivative values in this area to 0
```

```
      do 525 j=1,n+1
      f(m+1,j)=5.
      bda(j)=pix*sin(piy*y(j))
525   continue
      do 527 j=jpi,jpf
      bda(j)=0
527   f(1,j)=5.
      do 528 i=1,m+1
      bdc(i)=piy*sin(pix*x(i))
      bdd(i)=bdc(i)
528   continue
c
c  solve b*utilda = v
      call possol (0,x,m,mbdcnd,bda,bdb,y,n,nbdcnd,bdc,
     1                  bdd,elmbda,f,idimf,pertrb,ierror,w)
      write (2,530) ierror
530   format("ierror from possol = ",i5)
c
c  load beta with the rhs  v-a*u
      do 535 k=jpi,jpf
      beta(k-jpi+1)=5.-f(1,k)
535   continue
c
c  solve for beta
      call sol(jpf-jpi+1,20,c,beta,ip)
c
c  reload f with the modified v+beta*e
      do 540 i=1,m+1
      do 540 j=1,n+1
      f(i,j)=-(pix**2+piy**2)*sin(pix*x(i))*sin(piy*y(j))
540   continue
c
c  put in the fixed boundary at x=b
      do 542 j=1,n+1
      f(m+1,j)=5.
542   continue
c
c  put in the fixed boundary at x=a and add the beta value
      do 543 j=jpi,jpf
543   f(1,j)=5.+beta(j-jpi+1)
c
c  solve b*u=v+beta*e to get the final u
      call possol (0,x,m,mbdcnd,bda,bdb,y,n,nbdcnd,bdc,
     1                  bdd,elmbda,f,idimf,pertrb,ierror,w)
      write (2,545) ierror
545   format("ierror from possol = ",i5)
c
c  print out the results
      do 600 i=1,m+1
      write (2,550) (f(i,j),j=1,n+1)
550   format (4e12.5)
600   continue
c
c  print out the difference between the solution and the correct value
      write (2,650)
650   format("error values")
      do 800 i=1,m+1
      write (2,550) ((f(i,j)-5.*sin(pix*x(i))*sin(piy*y(j))),j=1,n+1)
```

```
165  800  continue
166 c
167       call exit(0)
168       end
```